

---

# Superword-Level Parallelism in the Presence of Control Flow

Jaewook Shin  
Mary Hall  
Jacqueline Chame

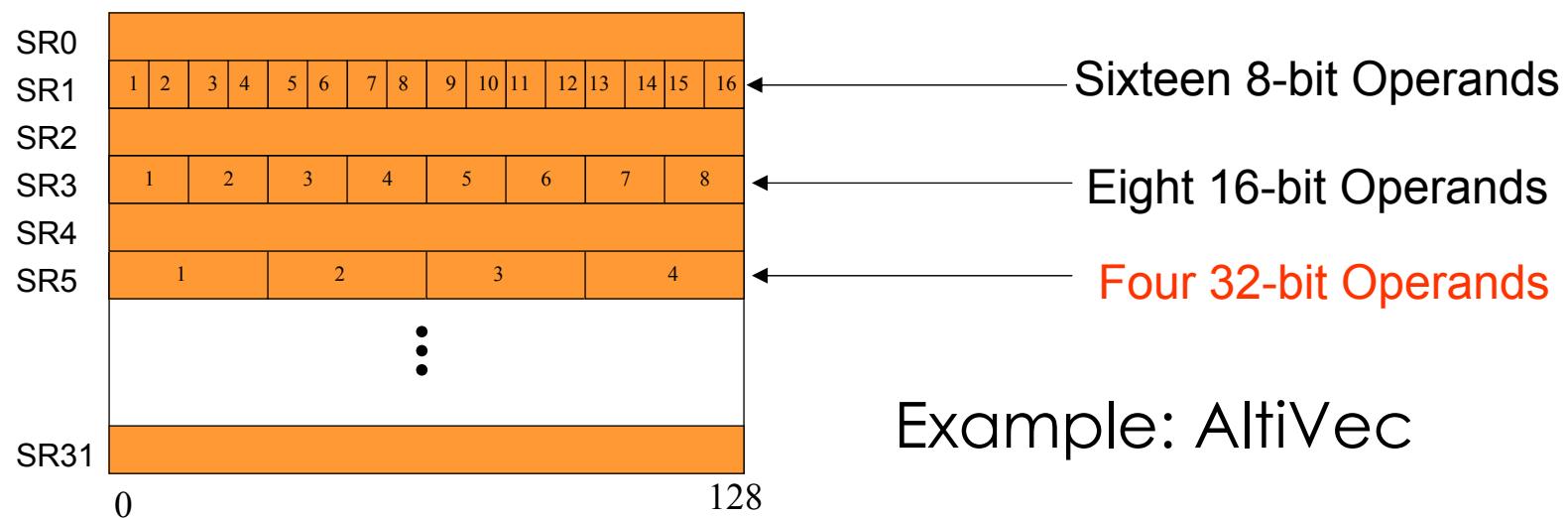
CGO'05



March 22 2005

# Multimedia Extension Architectures

- Multimedia applications are becoming increasingly important.
  - Most microprocessors have multimedia extensions.
    - SIMD parallelism
    - Variable-sized data fields



# Superword-Level Parallelism (SLP)

---

- Fine grain SIMD parallelism in aggregate data objects larger than a machine word

	SLP	Vector Parallelism
Strided Memory Access ?	NO	YES
Vector Length	$\leq 32$ elements	$\geq 64$ elements
Instruction Latency	$\sim 1$	1 per element
Aligned Memory Access ?	YES	NO

- Most compilers for multimedia extensions are based on conventional vectorization techniques.

# SLP Compiler (Larsen & Amarasinghe)

---

```
for (i=0; i<16; i++)
    a[i] = b[i] + c[i];
```



Unroll by 4

```
for (i=0; i<16; i+=4) {
    a[i+0] = b[i+0] + c[i+0];
    a[i+1] = b[i+1] + c[i+1];
    a[i+2] = b[i+2] + c[i+2];
    a[i+3] = b[i+3] + c[i+3];
}
```



Pack isomorphic statements

```
for (i=0; i<16; i+=4)
    a[i:i+3] = b[i:i+3] + c[i:i+3];
```

# Control Flow and the SLP Compiler

---

```
for (i=0; i<16; i++)
    if (a[i] != 0)
        b[i]++;
```



**Only parallelizes within a basic block !**

# Our Approach

---

```
for (i=0; i<16; i++)
    if (a[i] != 0)
        b[i]++;
```

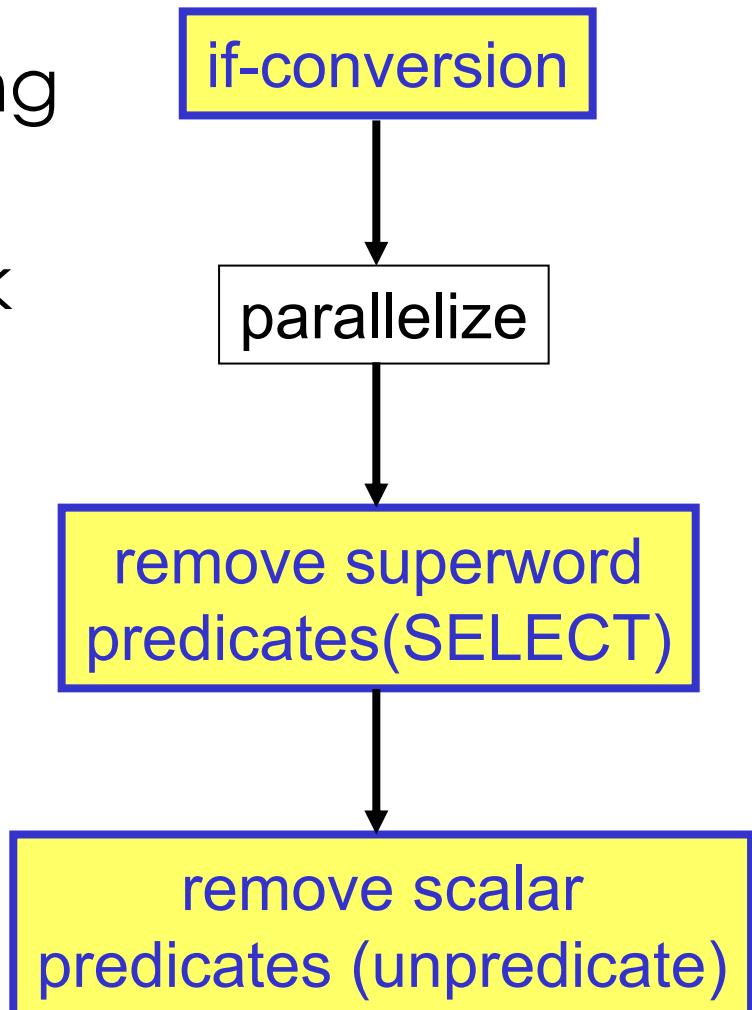


```
for (i=0; i<16; i+=4) {
    Vcond = a[i:i+3] != (0, 0, 0, 0);
    Vtemp = b[i:i+3] + (1, 1, 1, 1);
    b[i:i+3] = Combine b[i:i+3] and Vtemp
                according to Vcond;
}
```

# Key Concepts

---

- Borrow from optimizations for architectures supporting predicated execution
- Derive a large basic block of predicated instructions
- SELECT operations merge data values for different control flow paths
- Restore control flow



# If-Conversion

---

```
if (a != 0)
    b = b + 1;
```



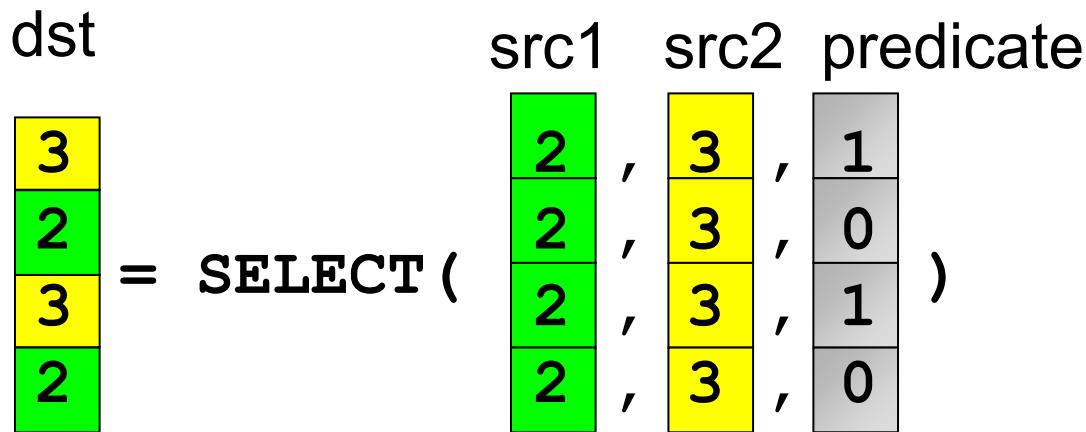
`pT = cond;  
pF = !cond;`

```
cond = a != 0;
pT, pF = pset(cond);
b = b + 1;           <pT>
```

`if (pT == true)
 b = b + 1;
else
 nop;`

# SELECT instruction

---



$V_a = V_b + (1, 1, 1, 1); \quad <vp>$

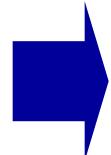


$V_{temp} = V_b + (1, 1, 1, 1);$   
 $V_a = \text{SELECT}(V_a, V_{temp}, V_p);$

# Unpredicate

---

```
bred[i] = fred;    <p>
bred[i] = 100;     <¬p>
bgre[i] = fgre;   <p>
bgre[i] = 100;    <¬p>
bblu[i] = fblu;  <p>
bblu[i] = 100;   <¬p>
```



```
if (p) {
    bred[i] = fred;
    bgre[i] = fgre;
    bblu[i] = fblu;
} else{
    bred[i] = 100;
    bgre[i] = 100;
    bblu[i] = 100;
}
```

# Algorithm

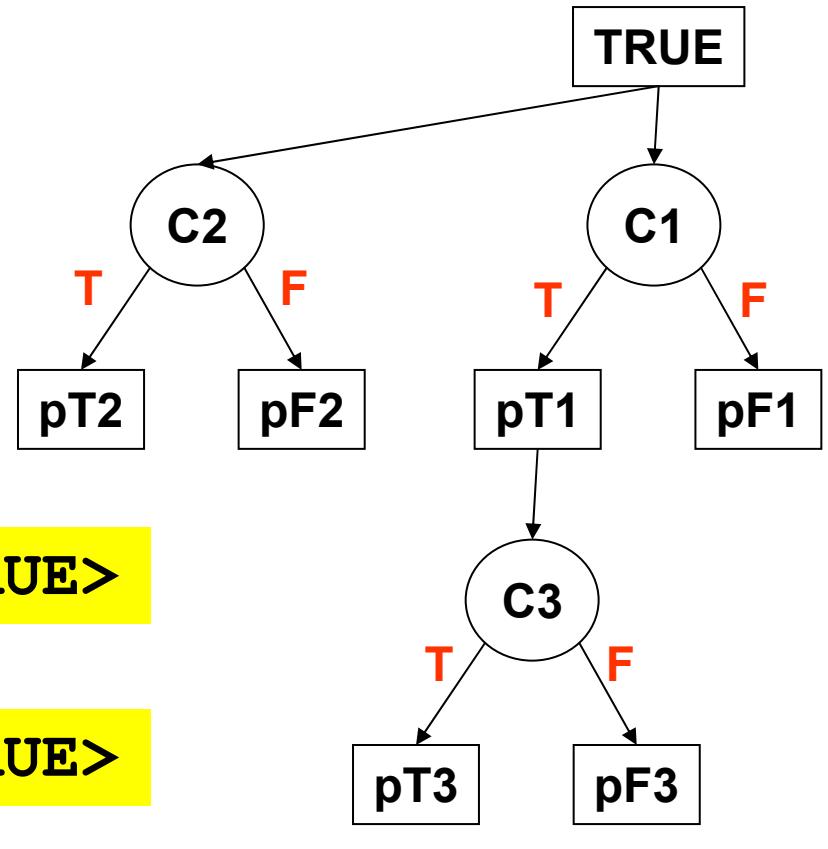
---

1. If-conversion:
  - Park and Schlansker's RK-algorithm
2. SELECT
  - Insert the minimum number of SELECT instructions
  - Use reaching definition based on predicate covering
3. Unpredicate
  - Try to reduce the number of conditional branches
  - Use predicate covering

# Predicate Hierarchy Graph (PHG)

- PHG represents relationships among predicates.

```
pT1, pF1 = pset (c1) ; <TRUE>
:
pT2, pF2 = pset (c2) ; <TRUE>
:
pT3, pF3 = pset (c3) ; <pT1>
:
```



# Predicate Covering

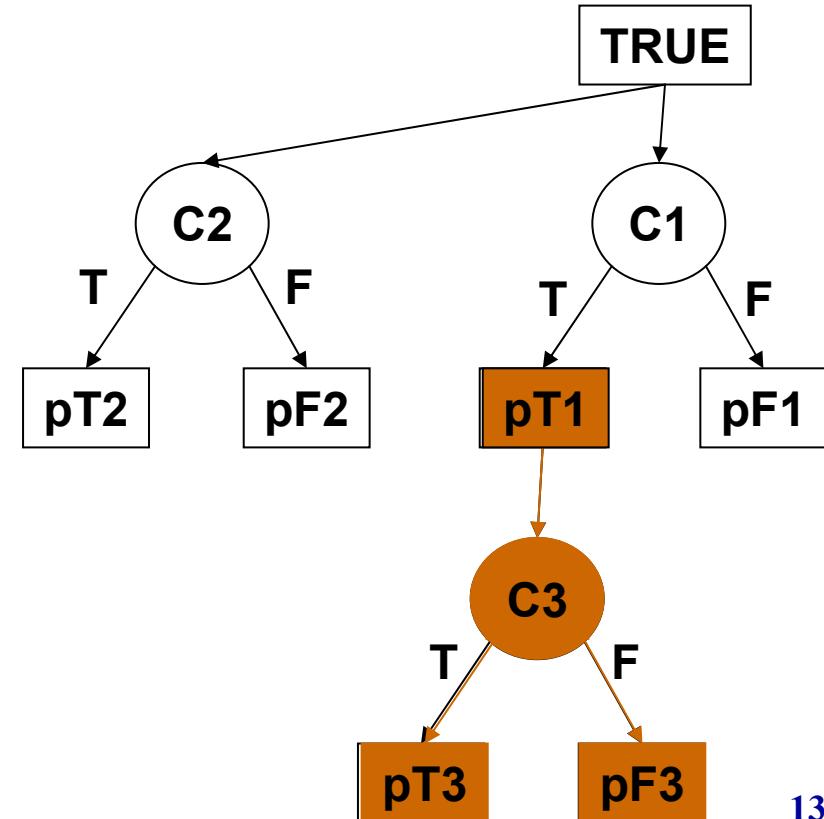
- A predicate  $p$  is covered by a set of predicates  $G$  if  $p = \text{true} \Rightarrow \exists p' \in G$  such that  $p' = \text{true}$ .

Q: Predicate covering  
predecessors of  $I3$  ?



⋮

$$G = \{ pT3, pF3, pT1 \}$$



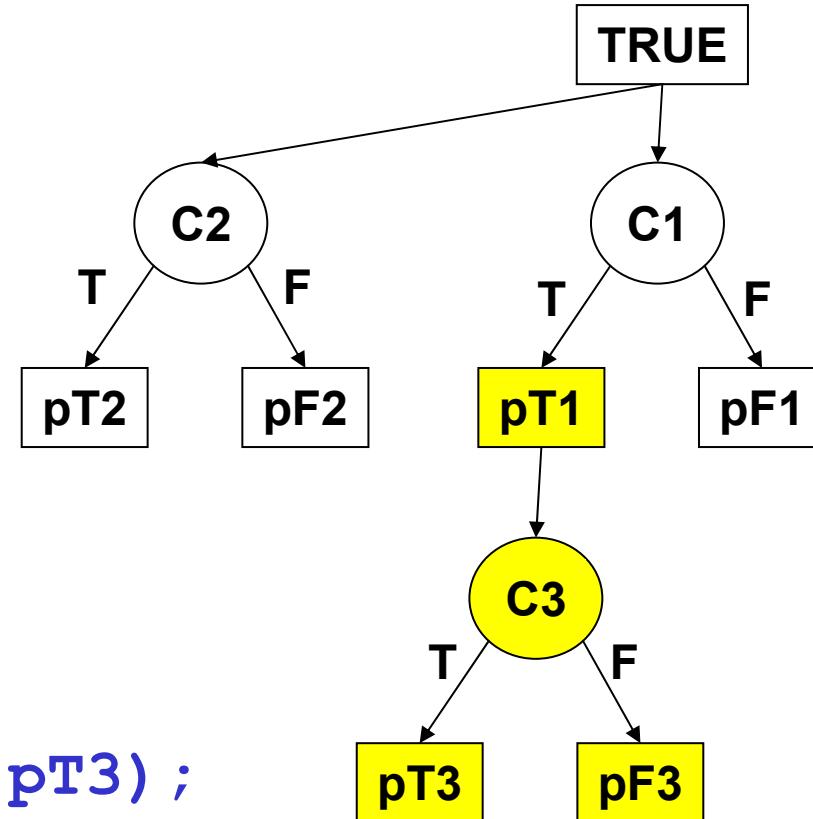
# SELECT Algorithm

- SELECT is not necessary for the first reaching definition

:  
d1:  $V_a = V_1 ; \langle pF3 \rangle$   
d2:  $V_a = V_2 ; \langle pT3 \rangle$   
u3:  $V_c = V_a ; \langle pT1 \rangle$



:  
d1:  $V_a = V_1 ;$   
d2:  $V_a = \text{SELECT} (V_a, V_2, pT3) ;$   
u3:  $V_c = V_a ;$

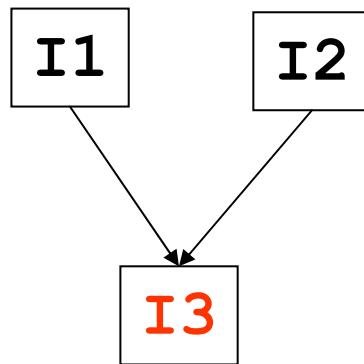


# Predicate CFG Generator

---

- Find the predicate covering predecessors for each predicated instruction.

I1 <pF3>  
I2 <pT3>  
**I3 <pT1>**



(a) predicated scalar code

(b) CFG

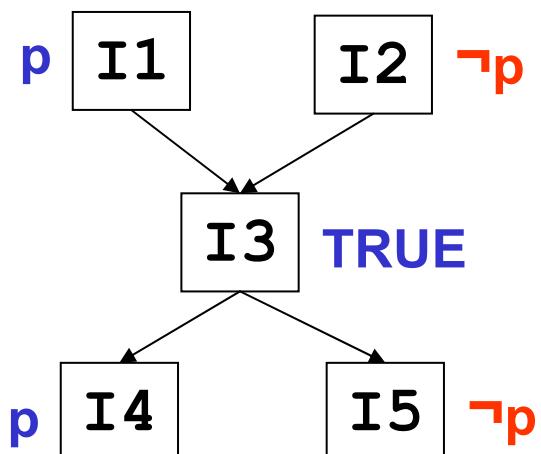
```
if(pT1) {  
    if(pF3)  
        I1;  
    else  
        I2;  
    I3;  
}
```

(c) code generated

# Unpredicate Algorithm

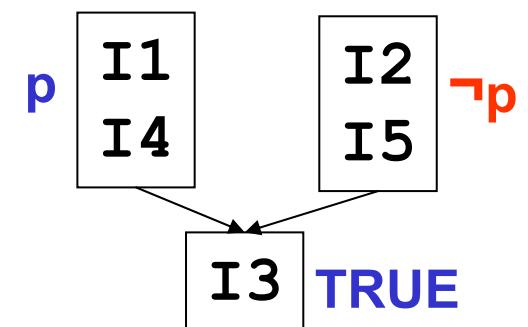
- Schedule each instruction within an existing basic block where it is safe.
- If no such basic block exists, use ‘predicate CFG generator’.

I1 <p>  
I2 <¬p>  
I3 <TRUE>  
I4 <p>  
I5 <¬p>



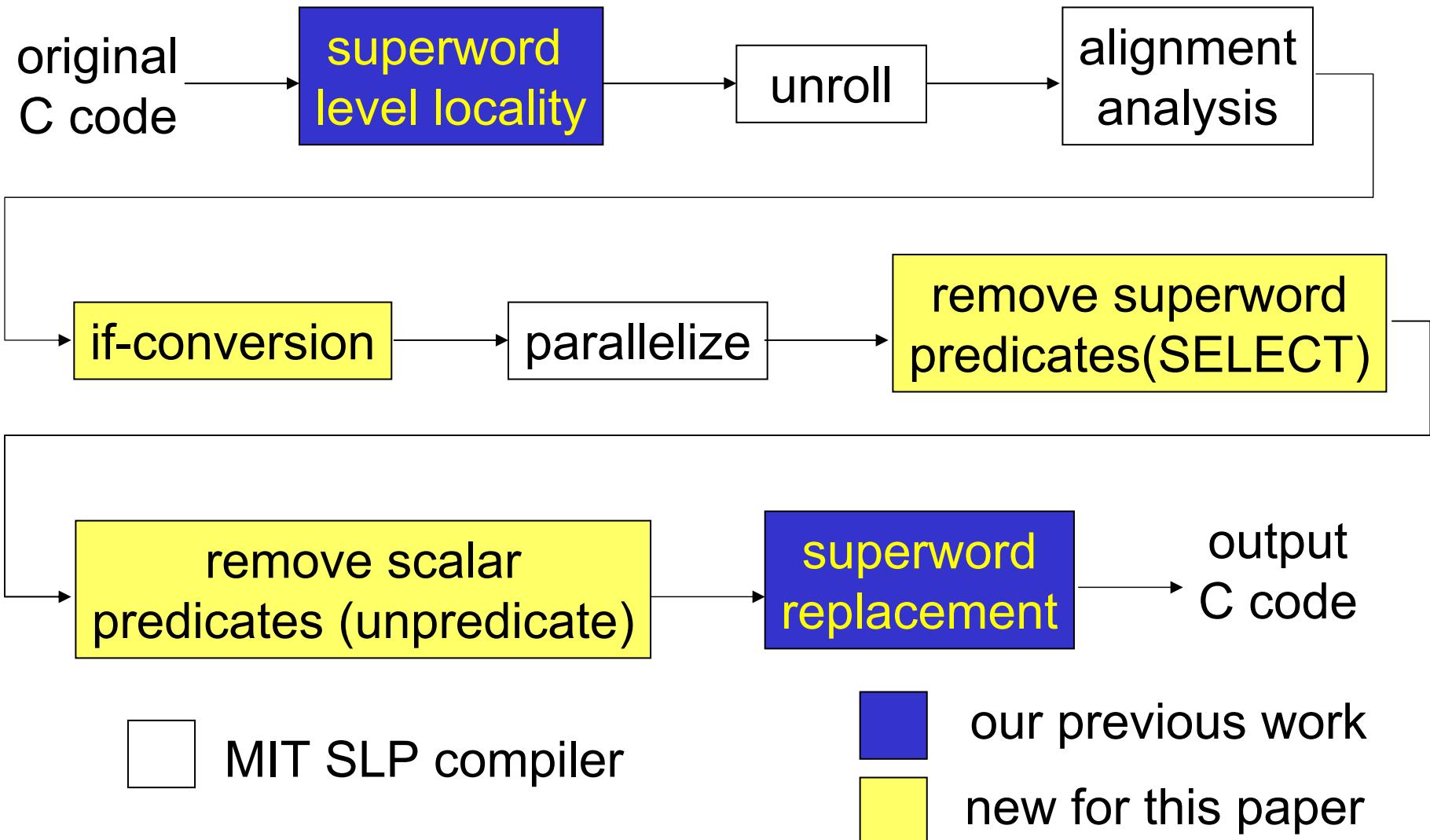
(a) predicated scalar code

(b) predicate CFG generator



(c) unpredicate

# Our Implementation



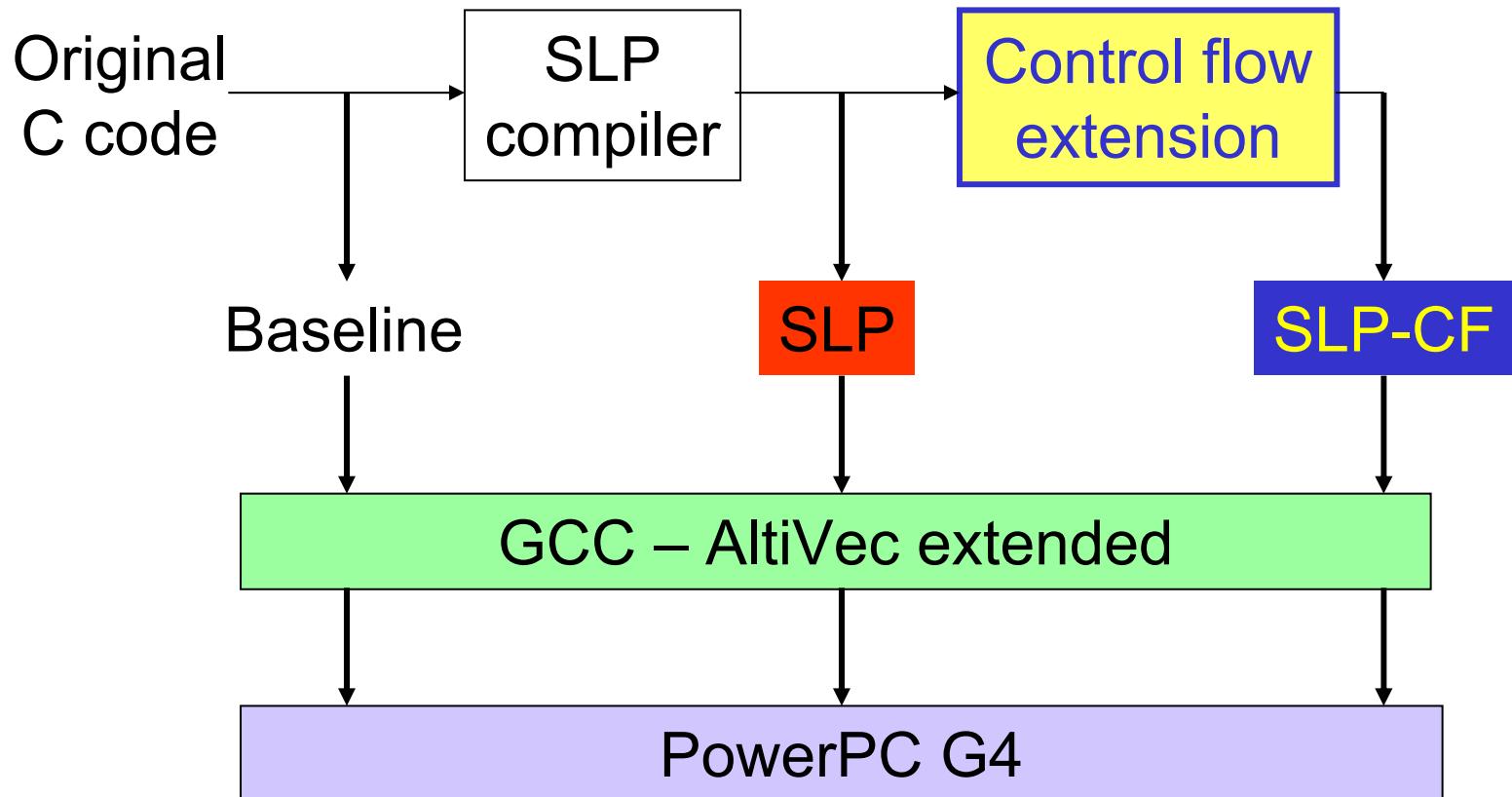
# Applications

---

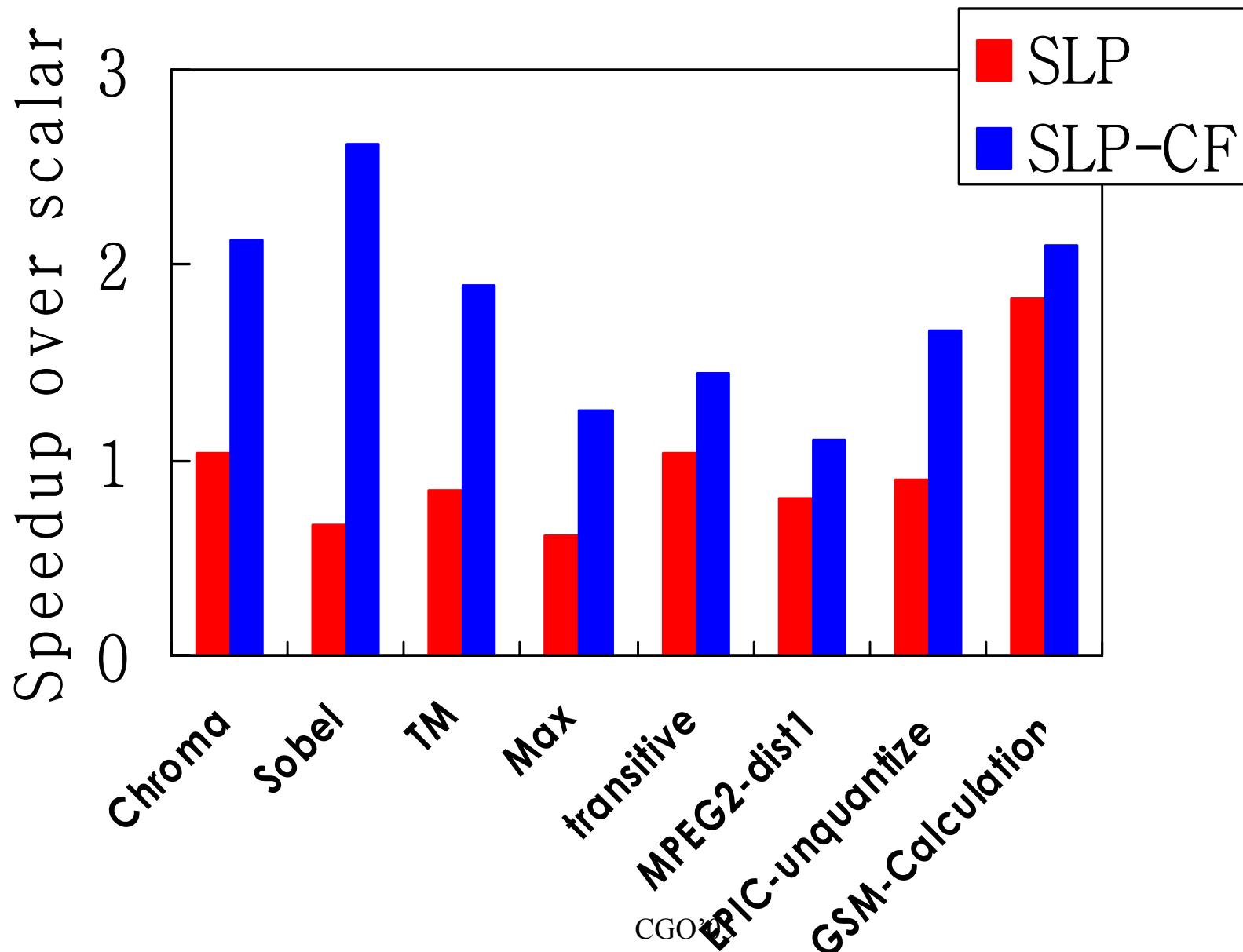
- Kernels
  - 1. Chroma: Chroma keying of two images
  - 2. Sobel: Sobel edge detection
  - 3. TM: Template Matching
  - 4. Max: Max value search
  - 5. transitive: Shortest path search
- Functions from UCLA MediaBench
  - 1. MPEG2-dist1: dist1 of MPEG2 encoder
  - 2. EPIC-unquantize: unquantize\_image of uneptic
  - 3. GSM-Calculation: Calculation\_of\_the\_LTP\_parameters of gsmencode
- Two data set sizes
  - 1. Large: Representative data set
  - 2. Small: Isolates parallelization effects

# Experimental Flow

---

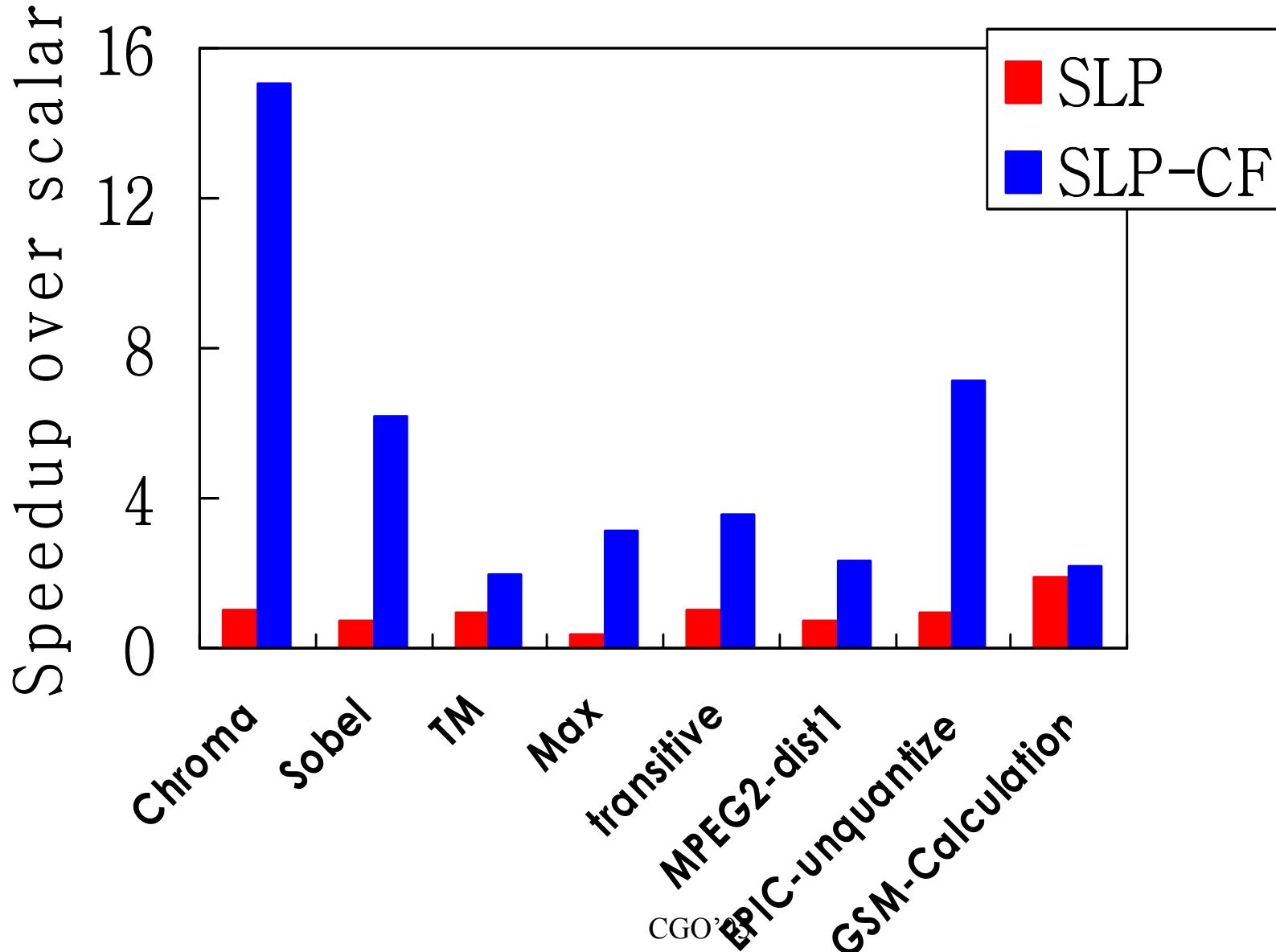


# Overall Improvements: large data



# Overall Improvements: small data

---



# Related Research

---

- Vectorization techniques for conditionals
  - Sreraman and Govindarajan(IJPP'00)
  - Bik et. al.(IJPP'02)
- Architectures with SELECT
  - Multimedia Extension Architectures: AltiVec
  - Processing-In-Memory: DIVA
  - Vector machines: Smith et. al.(ISCA'00)
- Phi-predication: Chuang et. al.(CGO'03)
  - Requires scalar SELECT
- Predicate CFG generator: Mahlke('96)

# Conclusion

---

- Compiler system to exploit SLP in the presence of control flow
- Compiler algorithms to
  - Minimum SELECTs
  - Restore efficient control flow
- In an experiment with 5 kernels and 3 benchmarks, the performance improved by 1.97X ~ 15.07X.